<u>Problem No. 39: Horse Shoe Scoring</u>

This algorithm works by taking the coordinates for each four throws of every turn and adding the appropriate score to the turns overall score. After the last turn in prints the turn number and score in a table.

First, we will initialize $R = 10$ (horseshoe radius) and $r = 1$ (post radius).

We input the A and B coordinates in a string, then extract them into 4 integer variables "x1" "y1" "x2" "y2". We calculate the distance between each point and the origins using $D = \sqrt{X^2 + Y^2}$ and store them so that $'Da'$ is the distance between 'a' and origins and $'Db'$ is the distance from 'b' to origins. It Keeps taking input from user until it encounters an empty line.

Then we test the three cases using relations between the two circles:

<u>1)Ringer:</u> First to see if the post is inside circle 'A' ; $Da < R + r$. But since it's only a semicircle we need to make sure it's in the right half, and for that we need to calculate the maximum distance between the center of the post and 'B' that is possible before it exits the half circle. That distance is $Db(\max) = \sqrt{(R + r)^2 + R^2}$. And so, the second condition for this case will be $Db < Db(\max)$.

<u>2)Toucher</u>: Here we can use the rule of intersection between two circles $R - r \leq Da \leq R + r$. And since it's a semicircle we must also add the $Db \leq Db(\max)$ condition (we added the equal in this case because Db can only equal $Db(\max)$ when the two circles are touching).

<u>3)Swinger:</u> Here we can use $Da > R + r$ to make sure the post is outside of the horseshoe. Then $Db < Db(\max)$ to make sure it's close enough that rotating about b with make the post intersect with the semicircle.

Each turn it calculates the score over 4 throws using the above conditions (ringer = 5 points, toucher = 2 points, swinger = 1 point). It then stores score in corresponding place in "score" array.

After it's done and all scores have been stored it prints them out in a table.

<u>Pseudo code:</u>

```
1. Define four float variables x1, y1, x2, y2 to store the coordinates of the horseshoe.
2. Define two float variables d1, d2 to store the distances between the center of the post and a
and b.
3. Initialize two float variables r1 and r2 to the radius of the horseshoe and post, respectively.
4. Initialize two float variables sum and diff to r1+r2 and r1-r2.
5. Define an integer array named 'score' of size 9999 to store the scores for each turn.
6. Define a character array named 'line' of size 100 to read input from the user.
7. Define two integer variables 'i' and 'j' to iterate over the turns and throws.
8. Define an integer variable 'count' to count the number of inputs read from the user.
9. Iterate over each turn until an empty line is entered:
    a. Iterate over each throw in the turn (four throws):
        i. Read input from the user and store it in 'line'.
        ii. Extract the coordinates of a and b from 'line' and store them in 'x1', 'y1', 'x2', 'y2'.
        iii. Calculate the distance between the center of the post and a and b and store them in 'd1' and 'd2'.
        iv. Check which of the three cases apply to the throw:
            1. If d1 < sum and d2<sqrt(sum^2+r1^2) , increment the score for the turn by 5.
            2. If d1 <= sum and d1 >= diff and d2<=sqrt(sum^2+r1^2), increment the score for the turn by 2.
            3. If d2<sqrt(sum^2+r1^2) and d1>sum, increment the score for the turn by 1.
    b. Store the score for the turn in the corresponding index of the 'score' array.
10. Print the turn number and score for each turn in a table.
```